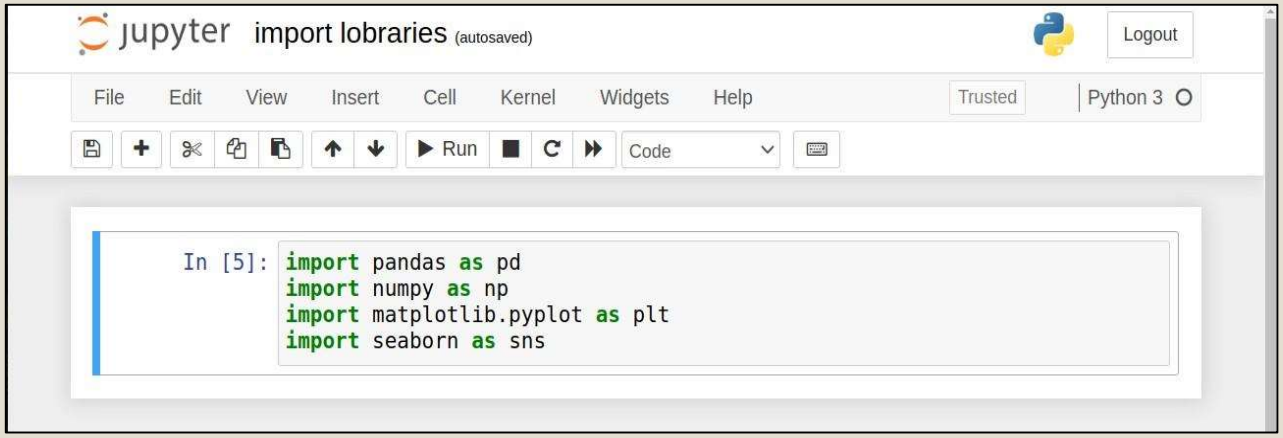
NUMPY

Installing Libraries

If you installed Anaconda, you do not need to download any libraries as it automatically installs all the popular data science libraries such as Pandas, Numpy, Matplotlib, Seaborn, etc.

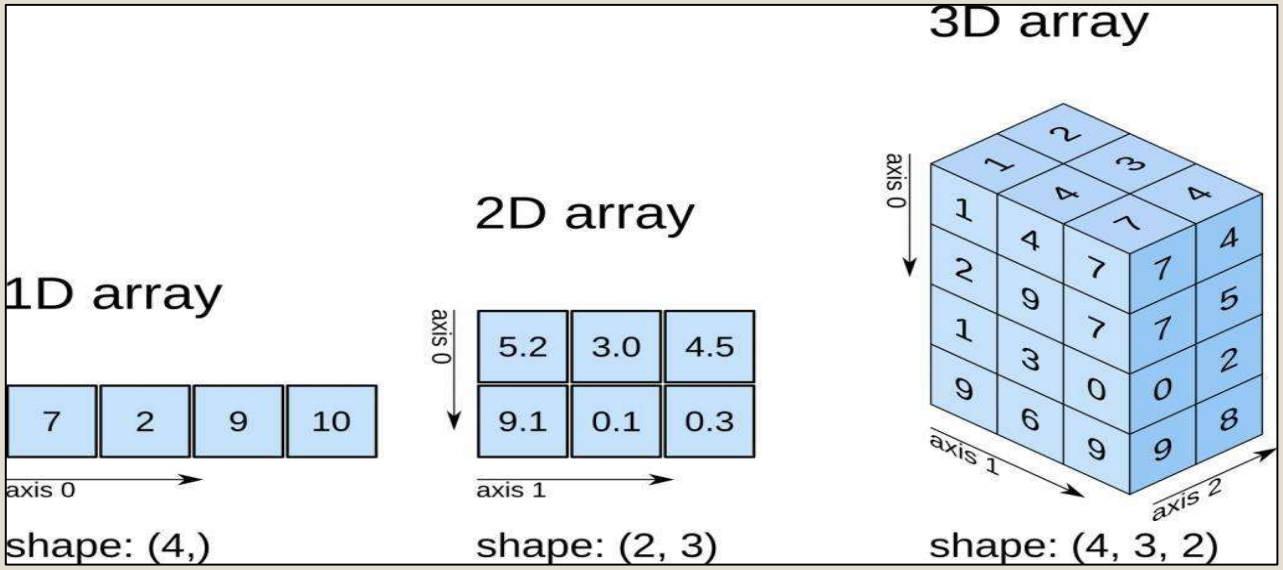
Importing Libraries

- Open your Jupyter Notebook.
- To import a library we use the keyword import followed by library name.
- We can use the as keyword to use abbreviations for our library names.
- The common abbreviations used are
  - pd for pandas
  - np for numpy
  - plt for matplotlib.pyplot
  - sns for seaborn



NumPy Library

- NumPy stands for Numerical Python.
- It provides a data structure called NumPy array, which is a grid of values.
- It also provides a collection of high-level mathematical functions which can be performed on multi-dimensional NumPy arrays.



NumPy Arrays

What are NumPy Arrays

- NumPy array is a multidimensional data structure designed to handle large data sets easily.
- A NumPy array is called **ndarray**.
- We can find the number of dimensions of a NumPy array using .ndim.

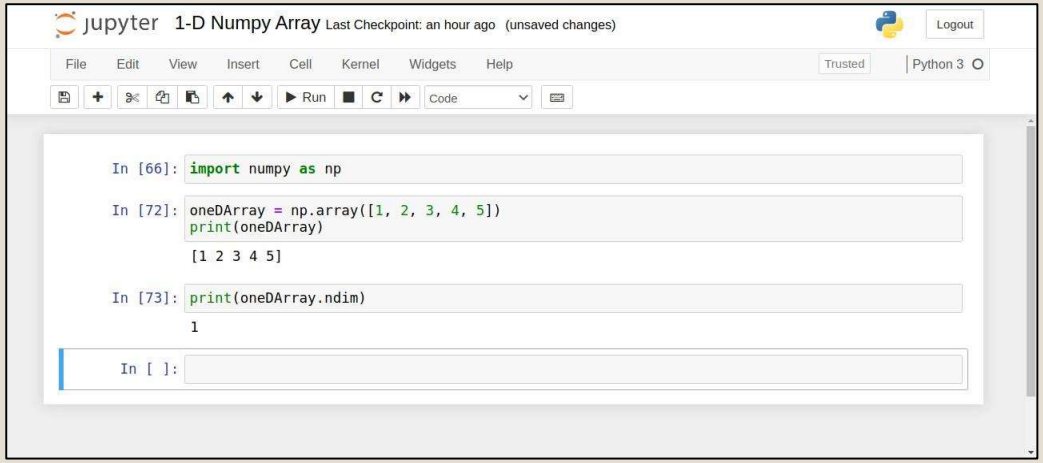
NumPy Arrays vs Python Lists

- NumPy arrays provide more built-in functionality as compared to Python lists.
- NumPy arrays make working with huge multi-dimensional data sets much easier with fewer syntax.
- NumPy arrays are also more efficient than Python lists in terms of memory consumption and speed.

Creating NumPy Arrays

1-D NumPy Arrays

- A 1-D NumPy array is where each element of the outermost array is a 0-D array (scalar).
- We can create a NumPy array using the array() function in the NumPy library.
- We can create a NumPy array using either Python lists or tuples.
- To create a 1-D NumPy Array, we provide a 1-D Python list or tuple to the array() function.



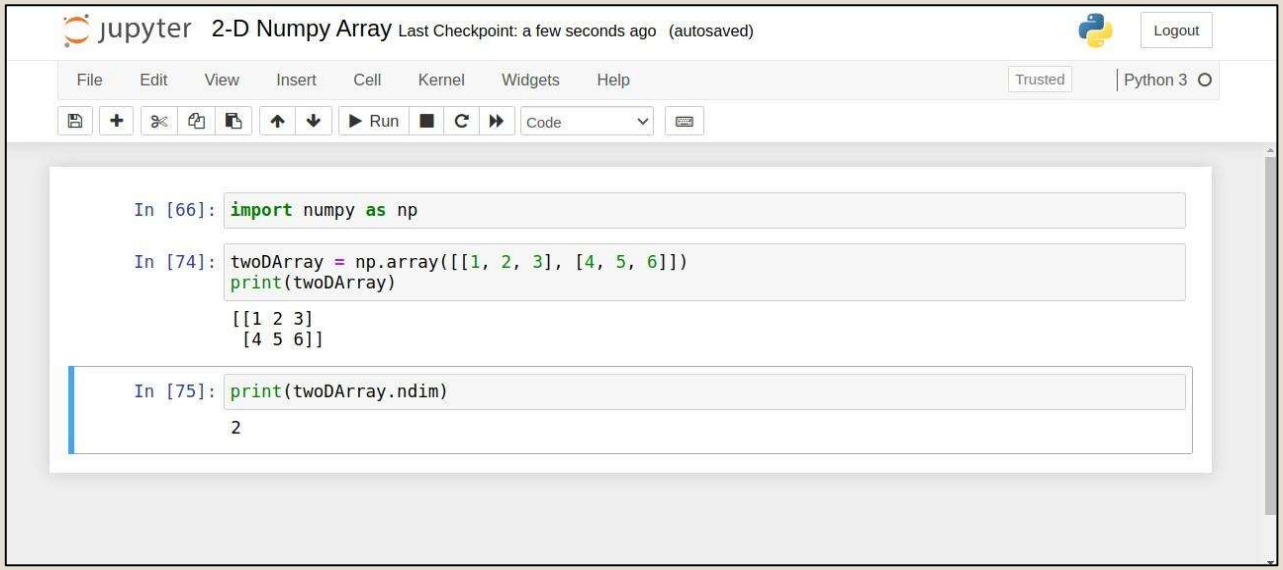
argument

```
In [66]: import numpy as np

In [72]: oneDArray = np.array([1, 2, 3, 4, 5])
         print(oneDArray)

         [1 2 3 4 5]

In [73]: print(oneDArray.ndim)

         1
```

Creating NumPy Arrays

2-D NumPy Arrays

- A 2-D NumPy array is where each element in the outermost array is a 1-D array.
- To create a 2-D NumPy Array, we provide a 2-D Python list or tuple to the array() function.



argument

```
In [66]: import numpy as np

In [74]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         print(twoDArray)

         [[1 2 3]
          [4 5 6]]
```

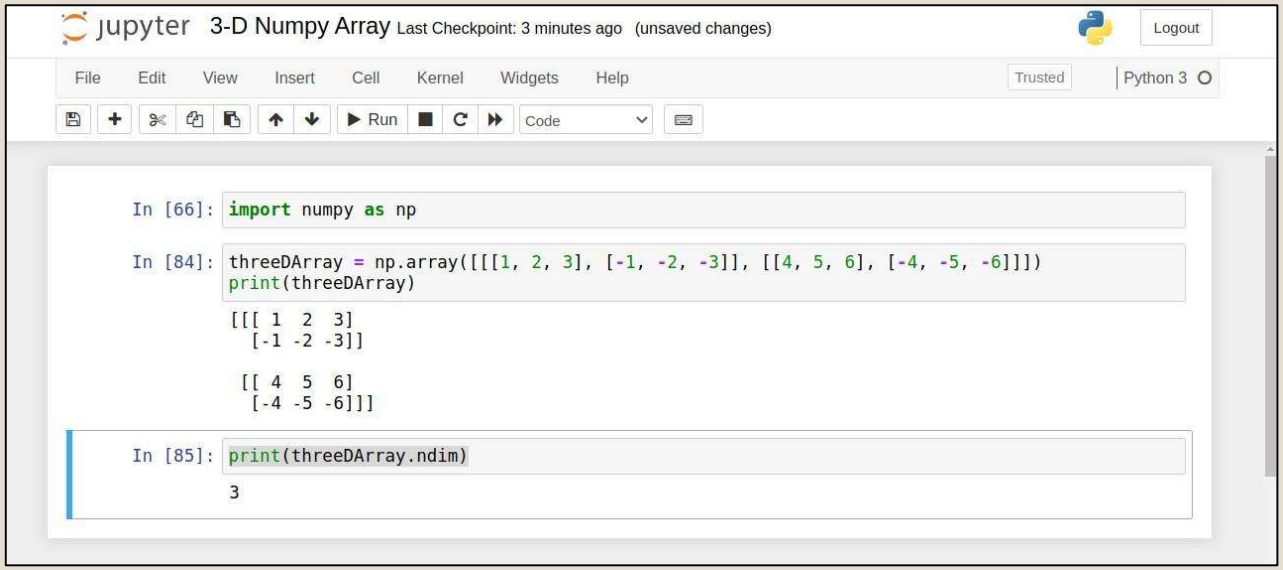
1D 1D

```
In [75]: print(twoDArray.ndim)

         2
```

Creating NumPy Arrays

3-D NumPy Arrays

- A 3-D NumPy array is where each element of the outermost array is a 2-D array.
- To create a 3-D NumPy Array, we provide a 3-D Python list or tuple to the array() function.



3D

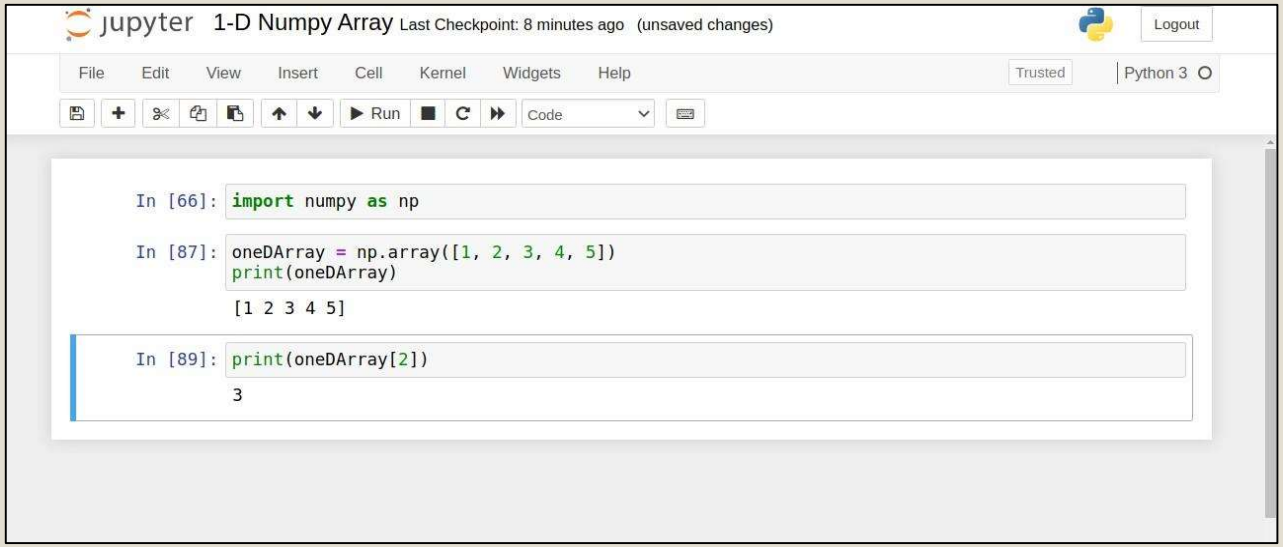
```
66]: import numpy as np

84]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
     print(threeDArray)

[[[ 1  2  3]
  [-1 -2 -3]]

 [[ 4  5  6]
  [-4 -5 -6]]]
```

2D 2D

Indexing NumPy Arrays

1-D NumPy Arrays

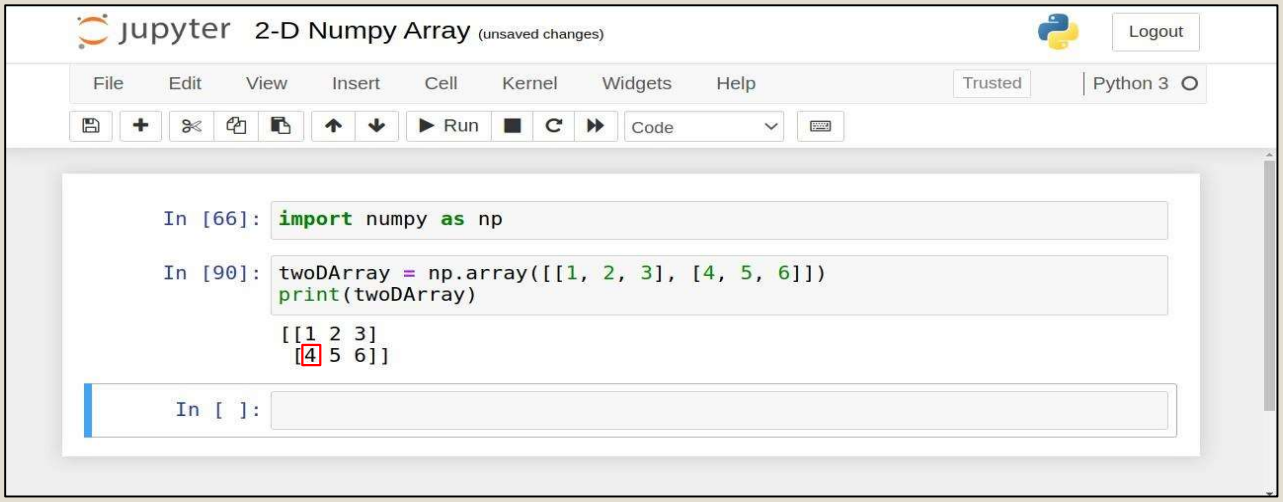
- Indexing a 1-D NumPy array is the same as indexing a 1-D Python list.
- Provide the index of the element inside the square brackets to get that element.



Indexing NumPy Arrays

2-D NumPy Arrays

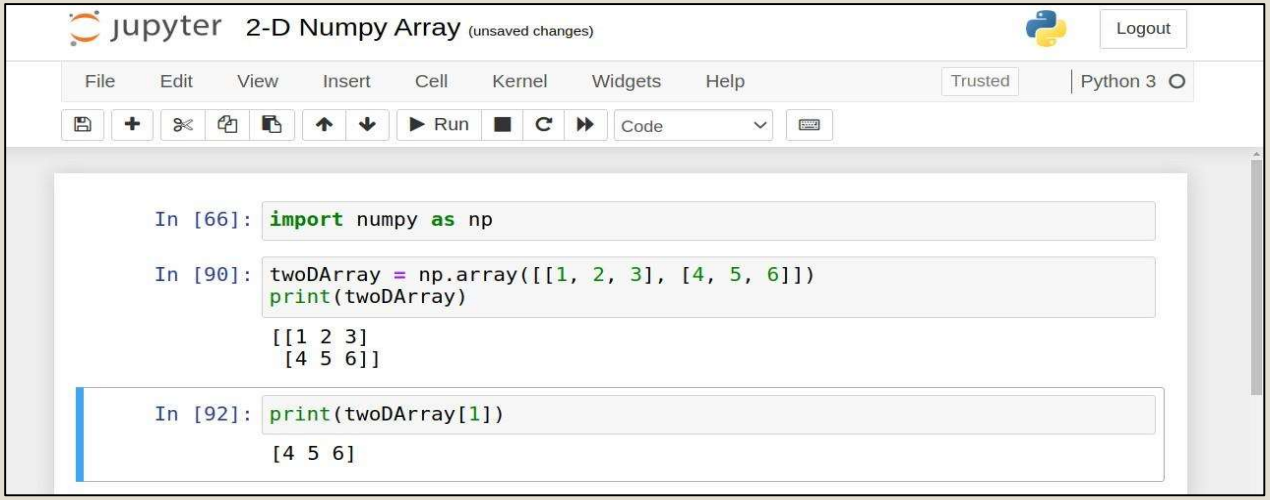
- To index a 2-D NumPy array, we provide 2 values inside the square brackets ([ ]).
  - First value is the index of the inner array
  - Second value is the index of the element inside the inner array
- In the following example, we get the first element of the second array.



Indexing NumPy Arrays

2-D NumPy Arrays

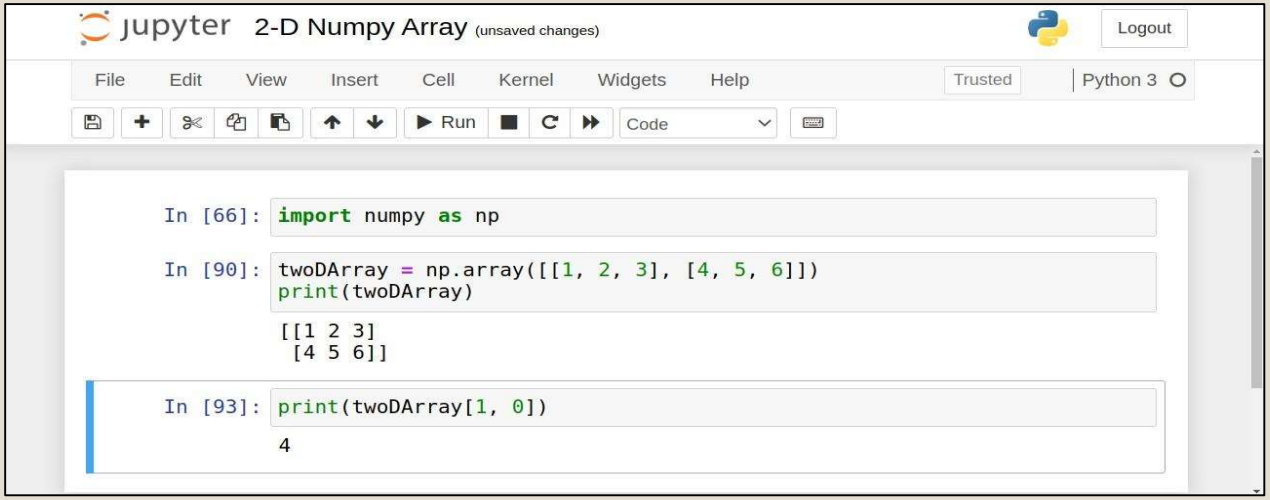
- The first dimension contains 2 arrays.
- If we say twoDArray[1], we get the second array.



Indexing NumPy Arrays

2-D NumPy Arrays

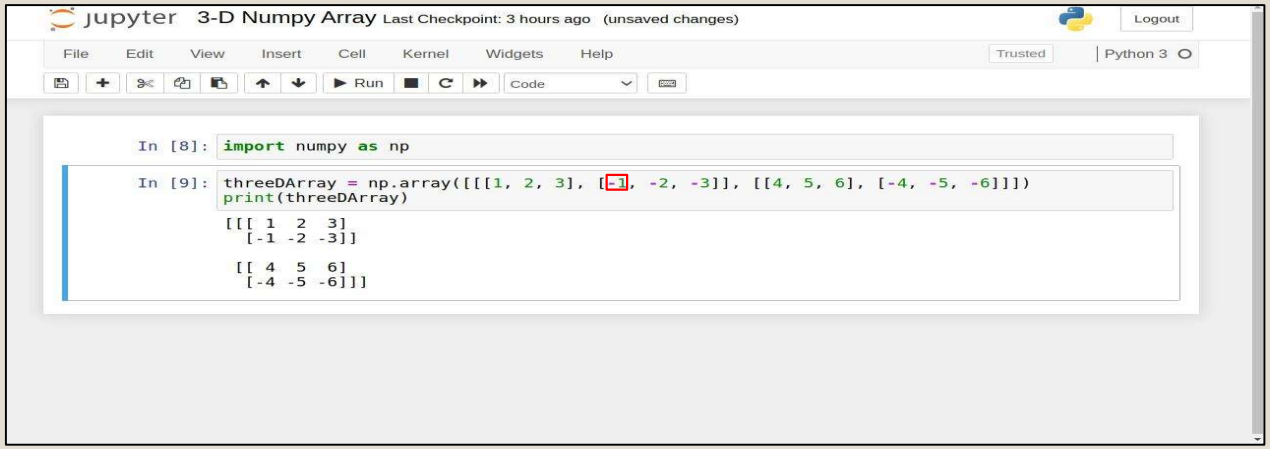
- The second dimension contains 3 elements.
- If we say twoDArray[1, 0], we get the first element of the second array.



Indexing NumPy Arrays

3-D NumPy Arrays

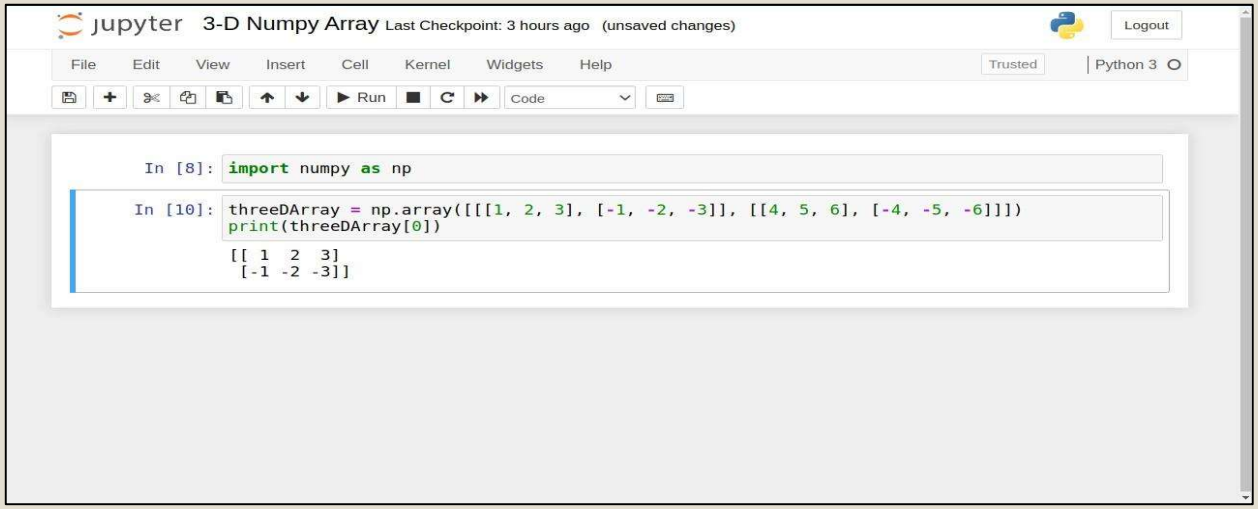
- To index a 3-D NumPy array, we provide 3 values inside the square brackets ([ ]).
  - First value is the index of the inner 2-D array in the first dimension.
  - Second value is the index of the inner 1-D array in the second dimension.
  - Third value is the index of the element in the third dimension.
- In the following example, we get the first element of the second array of the first array.



Indexing NumPy Arrays

3-D NumPy Arrays

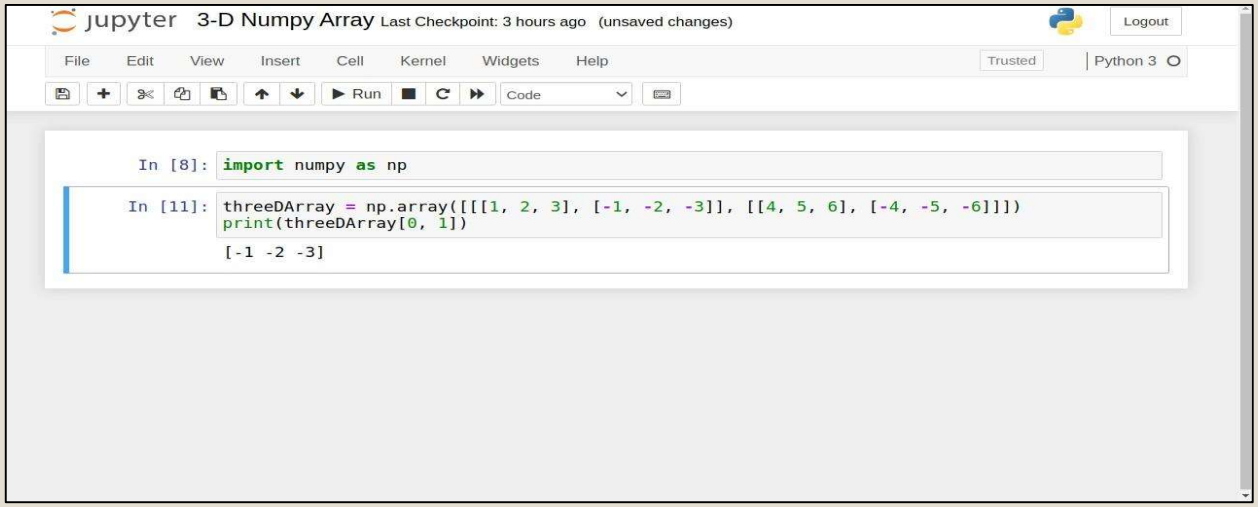
- The first dimension contains 2 arrays.
- If we say threeDArray[0], we get the first array.



Indexing NumPy Arrays

3-D NumPy Arrays

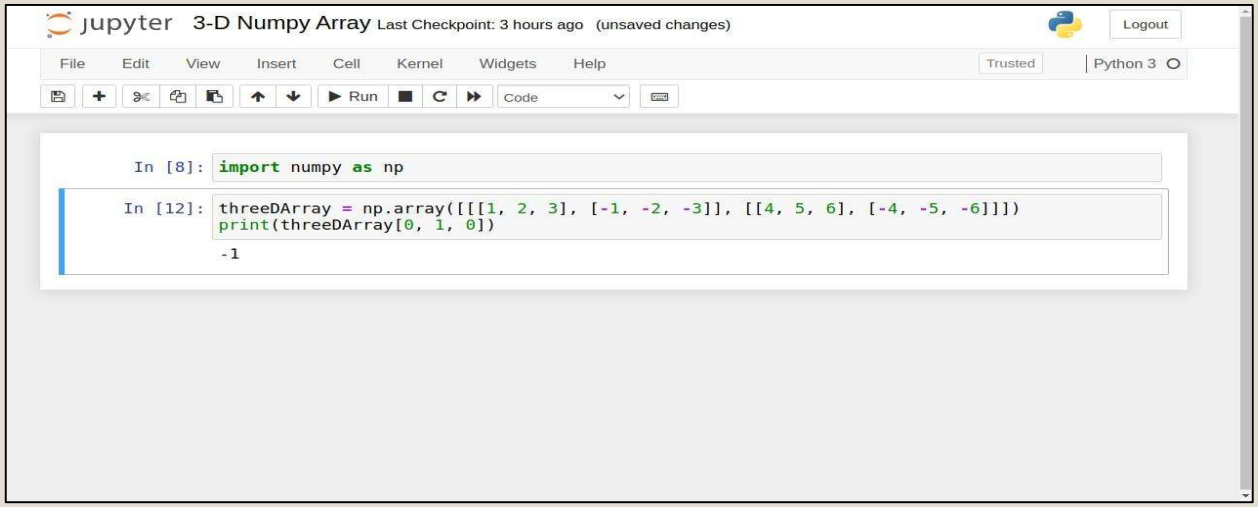
- The second dimension again contains 2 arrays.
- If we say threeDArray[0, 1], we get the second array of the first array.



Indexing NumPy Arrays

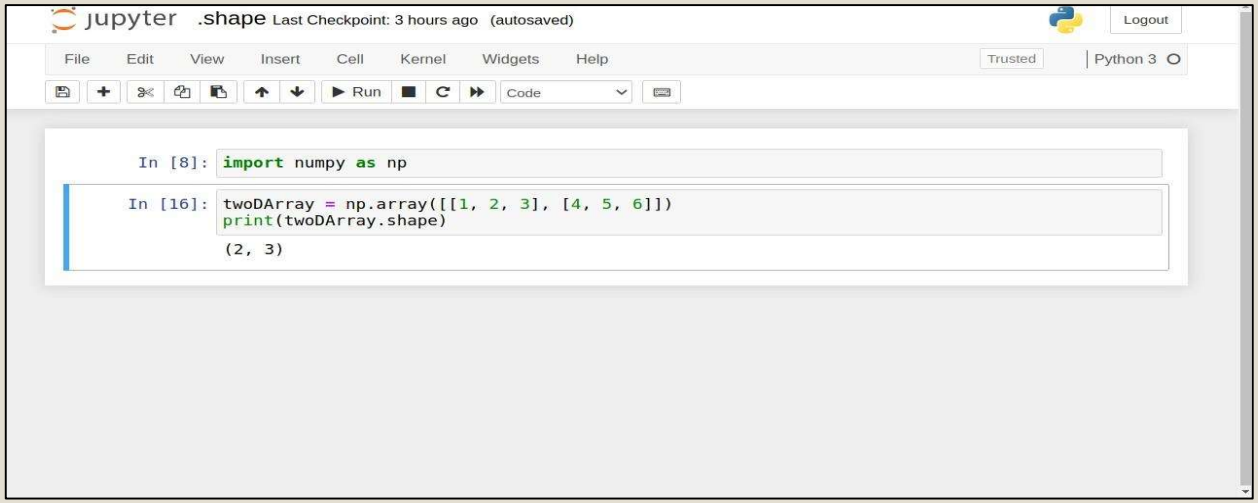
3-D NumPy Arrays

- The third dimension contains 3 values.
- If we say threeDArray[0, 1, 0], we get the first element of the second array of the first array.



Array Shape

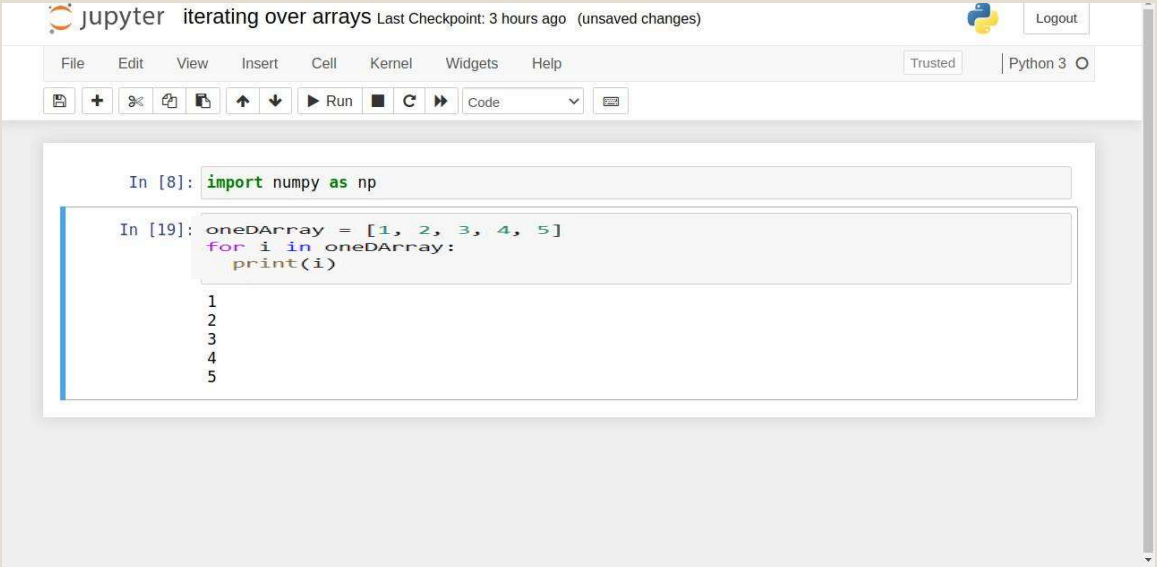
- NumPy arrays have a shape attribute which returns a tuple.
  - First value of the tuple gives the number of dimensions in the array
  - Second value of the tuple gives the number of elements in each dimension



Iterating Over NumPy Arrays

1-D NumPy Arrays

- We can use a for loop to iterate over a 1-D array just as we do in a 1-D Python list.



Iterating Over NumPy Arrays

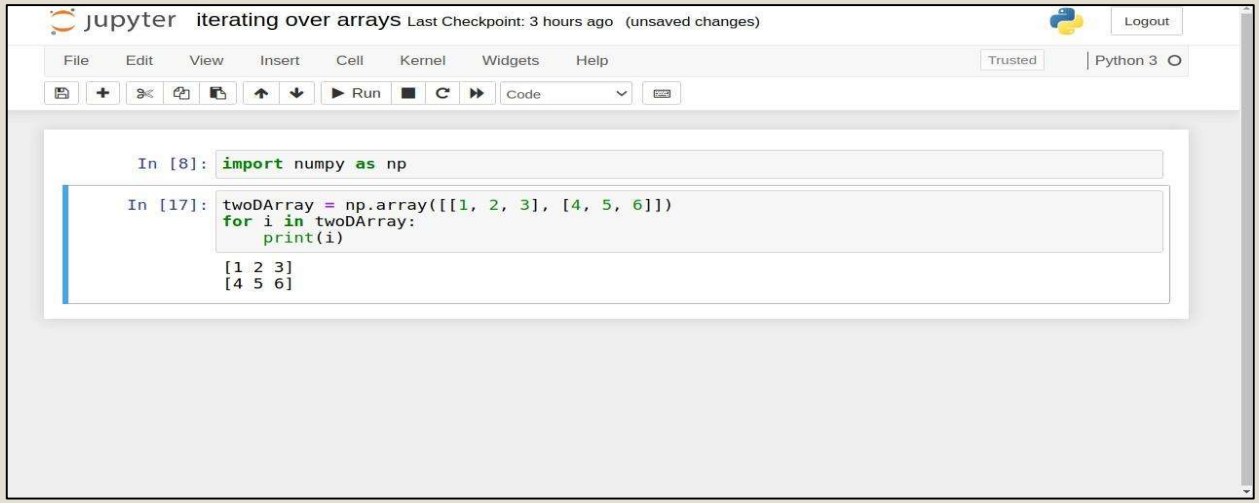
2-D NumPy Arrays

- We can use a nested for loop to iterate over a 2-D array.
  - The outer for loop iterates over the outer array.
  - The inner for loop iterates over the inner array.

Iterating Over NumPy Arrays

2-D NumPy Arrays

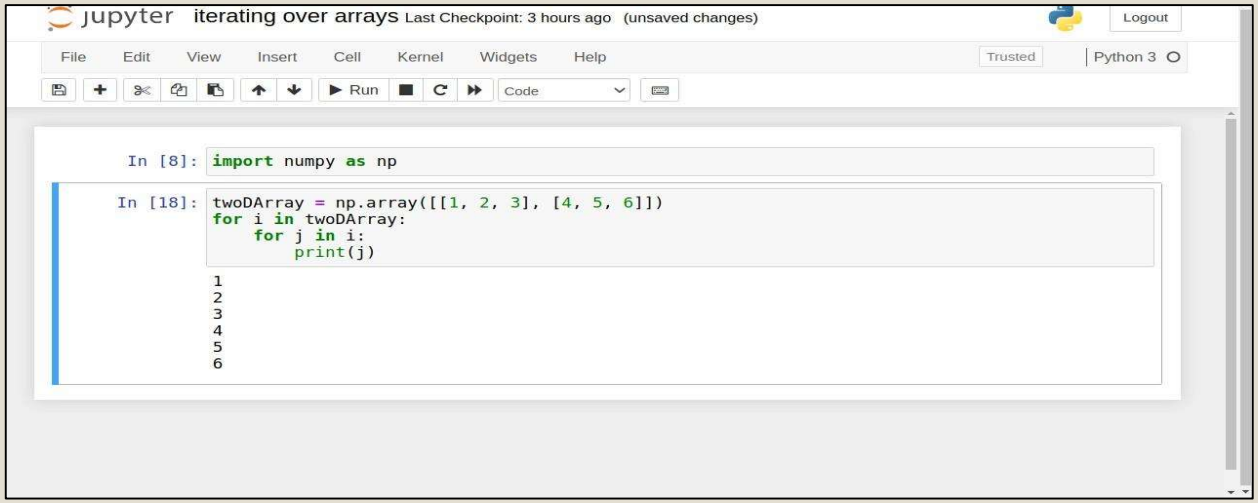
- We use a for loop to iterate over the outer array.
- We print all the inner arrays.



Iterating Over NumPy Arrays

2-D NumPy Arrays

- We use another for loop nested inside the outer for loop to iterate over the inner array.
- We print all the elements in each of the inner arrays.



Iterating Over NumPy Arrays

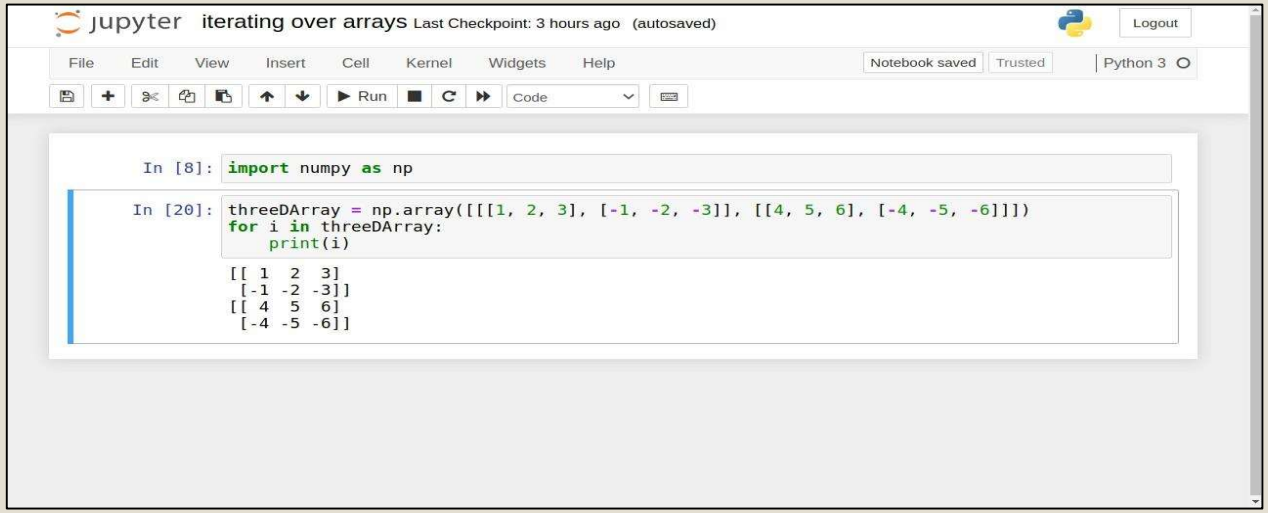
3-D NumPy Arrays

- We can use 3 nested for loops to iterate over a 3-D array.
  - The outermost for loop iterates over the arrays in the first dimension.
  - The middle for loop iterates over the arrays in the second dimension.
  - The innermost for loop iterates over all the elements in the third dimension.

Iterating Over NumPy Arrays

3-D NumPy Arrays

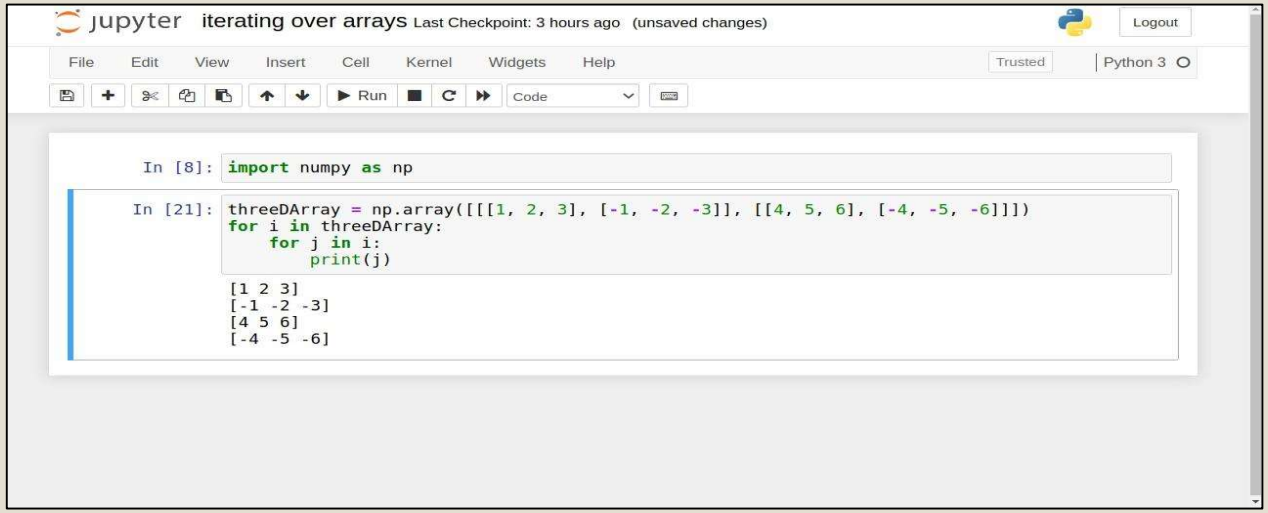
- Outermost array contains 2 arrays both of which are 2-D.
- We use a for loop to print these 2-D arrays.



Iterating Over NumPy Arrays

3-D NumPy Arrays

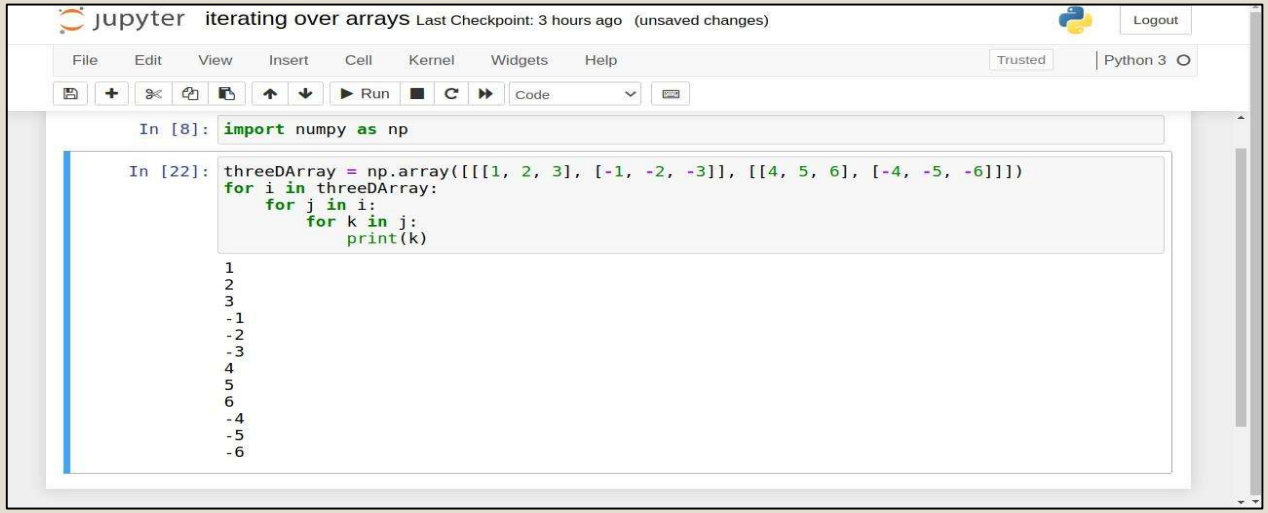
- Each of the 2-D array contains 2 arrays in the second dimension, each of which is 1-D.
- We use another for loop nested within the first for loop to print these 1-D arrays.



Iterating Over NumPy Arrays

3-D NumPy Arrays

- Each of the 1-D array contains 3 elements in the third dimension, each of which is 1-D.
- We use another for loop nested within the first 2 for loops to print these elements.

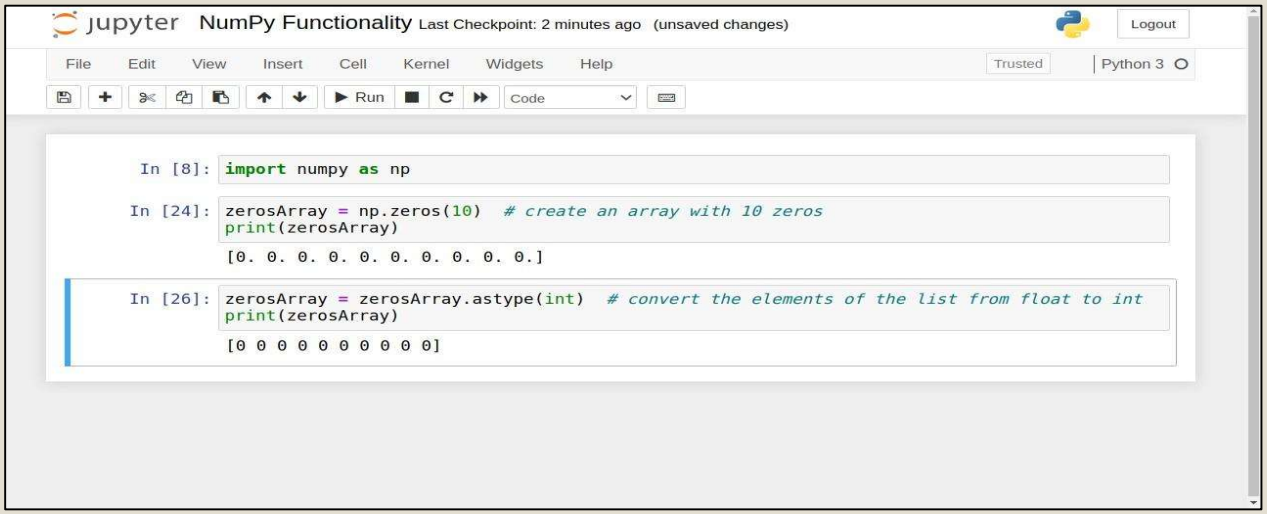


Mathematics for Data Science

- NumPy provides us with a huge collection of high-level functions for multi-dimensional arrays.
- Let’s have a look at some of the functionality provided by NumPy.

.zeros()

- To create a NumPy array prefilled with zeros, we can use the .zeros() built-in NumPy function.
- .zeros() gives us a list prefilled with float zeros. To convert this list into integer list, we use the .astype() function.

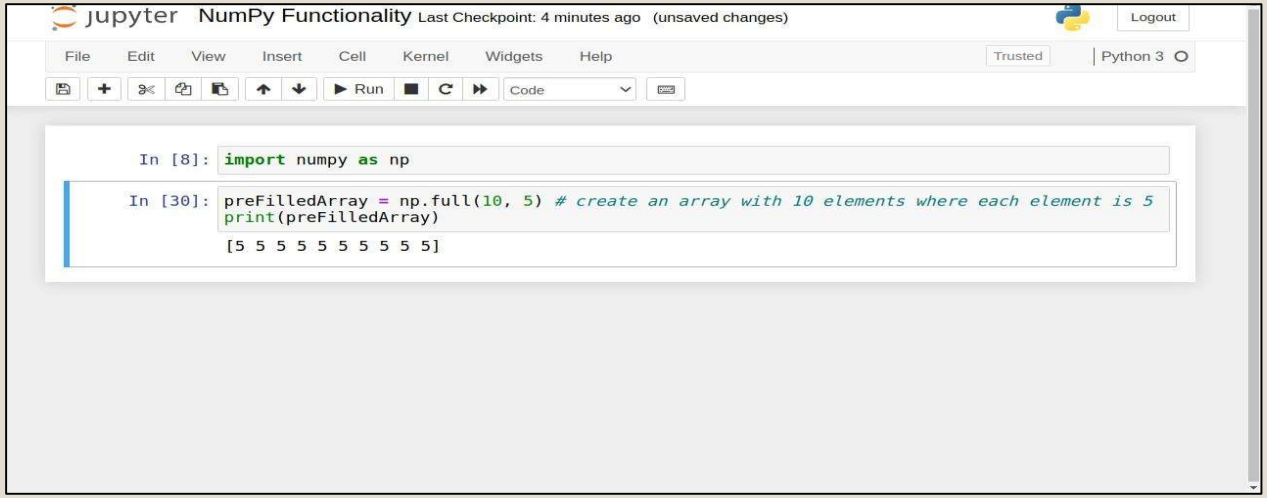


.ones()

- To create a NumPy array prefilled with ones, we can use the .ones() built-in NumPy function.
- .ones() gives us a list prefilled with float ones. To convert this list into integer list, we use the .astype() function.

.full()

- To create a NumPy array prefilled with some specific number, we can use the .full() built-in NumPy function.
  - First argument in the .full() function is the size of the array
  - Second argument in the .full() function is the value that we want our list to be pre-filled with



Scalar Operations

Addition

- We can add a scalar to a NumPy array simply by using the (+) operator.
- The scalar quantity is added to each of the elements of the array.
- Note that adding a scalar to a Python list will result in an error.

Scalar Operations

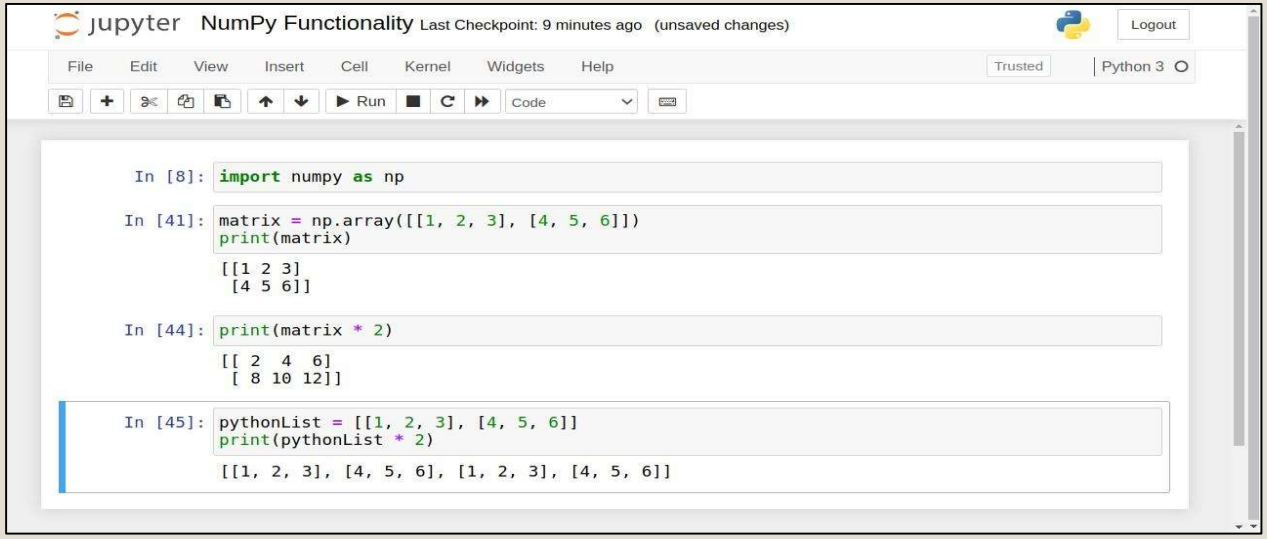
Subtraction

- We can subtract a scalar from a NumPy array simply by using the (-) operator.
- The scalar quantity is subtracted from each of the elements of the array.
- Note that subtracting a scalar from a Python list will result in an error.

Scalar Operations

Multiplication

- We can multiply a scalar with a NumPy array simply by using the (*) operator.
- The scalar quantity is multiplied with each of the elements of the array.
- Note that multiplying a scalar with a Python list will result in list concatenation.



Scalar Operations

Division

- We can divide a NumPy array by a scalar simply by using the (/) operator for float division or (//) operator for integer division.
- Each of the elements of the array is divided by the scalar.
- Note that dividing a Python list by a scalar will result in an error.

Scalar Operations

Power

- We can raise each element of a NumPy array to a power simply by using the (**) operator.
- Note that raising the elements of a Python list using (**) operator will result in an error.

Transpose

- We can take the transpose of a NumPy array by putting .T at the end of the array.
- Note that taking transpose of a Python list will result in an error.